PATENT
5181-95901
P6691

"EXPRESS MAIL" MAILING LABEL NUMBER
EV318248079US
DATE OF DEPOSIT 4/9/04
I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 C.F.R. 1.10 ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER FOR PATENTS, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450.

Derrick Brown

A MECHANISM AND METHOD FOR CACHE SNOOP FILTERING

By:

Robert E. Cypher

Atty. Dkt. No.: 5181-95901

Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Ph: (512) 853-8800

This application claims the benefit of U.S. Provisional Application No. 60/520,620 filed November 17, 2003.

BACKGROUND OF THE INVENTION

Field of the Invention

[0001] This invention relates to computer systems including processors that employ cache memory subsystems and, more particularly, to reducing the number of snoop transactions to a cache memory using a snoop filter.

Description of the Related Art

[0002] A cache memory is a high-speed memory unit interposed in the memory hierarchy of a computer system between a slower system memory and a processor. A cache typically stores recently used data to improve effective memory transfer rates to thereby improve system performance. The cache is usually implemented by semiconductor memory devices having speeds that are comparable to the speed of the processor, while the system memory utilizes a less costly, lower speed technology.

[0003] A cache memory typically includes a plurality of memory locations that each stores a block or a "line" of two or more words. Each line in the cache has associated with it an address tag that is used to uniquely identify the address of the line. The address tags are typically included within a tag array memory device. Additional bits may further be stored for each line along with the address tag to identify the coherency state of the line.

[0004] A processor may read from or write directly into one or more lines in the cache if the lines are present in the cache and if the coherency state allows the access. For example, when a read request originates in the processor for a new word, whether data or instruction, an address tag comparison is made to determine whether a valid copy of the requested word resides in a line of the cache memory. If the line is present, a cache "hit" has occurred and the data is used directly from the cache. If the line is not present, a cache "miss" has occurred and a line containing the requested word is retrieved from the system memory and may be stored in the cache memory. The requested line is simultaneously supplied to the processor to satisfy the request.

[0005] Similarly, when the processor generates a write request, an address tag comparison is made to determine whether the line into which data is to be written resides in the cache. If the line is present, the data may be written directly into the cache (assuming the coherency state for the line allows for such modification). If the line does not exist in the cache, a line corresponding to the address being written may be allocated within the cache, and the data may be written into the allocated line.

[0006] Because two or more copies of a particular piece of data can exist in more than one storage location within a cache-based computer system, coherency among the data is necessary. Various coherency protocols and specialized bus transfer mechanisms may be employed for this purpose depending on the complexity of the system as well as its requirements. For example, coherence between the cache and the system memory during processor writes may be maintained by employing either a "write-through" or a "write-back" technique. The former technique guarantees consistency between the cache and the system memory by writing the same data to both locations. The latter technique handles coherency by writing only to the cache, and by marking the entry in the cache as being modified. When a modified cache entry is later removed during a cache replacement

cycle (or is required by a device other than the processor), the modified data is typically written back to the system memory (and/or provided to the requesting device).

[0007] In a multiprocessor shared-memory computer system, separate caches associated with each of the processors may simultaneously store data corresponding to the same memory location. Thus, memory coherency within such systems must typically be handled using somewhat more elaborate and complex schemes. For example, coherency in multiprocessor shared-memory systems may be maintained through employment of either a directory-based protocol or a snooping protocol. In a directory-based protocol, a directory is maintained that indicates which processors have copies of each cache line. This directory is used to limit the processors that must monitor, and possibly respond to, a given request for a cache line. The use of directories reduces snoop traffic and thus allows larger systems to be built. However, the use of directories typically increases the system's latency (which is caused by the directory lookup), as well as the system's hardware complexity and cost.

[0008] In a snooping protocol, each processor broadcasts all of its requests for cache lines to all other processors. In many systems, this may be done through a common shared bus. The cache associated with each processor stores along with its address tags coherency information indicating the state of each of its stored lines. Each processor snoops the requests from other processors and responds accordingly by updating its cache tags and/or by providing the data. Thus, each request from another processor may require that a given processor access its own cache's tags to determine if the line exists within the cache, and to update the tag and/or provide the data if necessary. In systems that store cache tags off-chip, the rate at which these cache tags can be accessed may put a limit on the rate at which snoops can be processed and may further limit the bandwidth available for the processor's own cache transactions.

[0009] One solution to this problem is to store the cache tags on-chip (on the same chip as the processor), even for cache lines that are stored off-chip. However, this solution suffers from several drawbacks, including the large amount of processor area that must be devoted to maintain these cache tags, the lack of flexibility in changing off-chip cache sizes and organizations, and an increased latency when the data is present in the off-chip cache.

[0010] Regardless of whether the cache tags are stored on-chip or off-chip, snoop operations may be necessary to determine whether a remotely requested cache line exists in the cache, whether the cache line state, as reflected in the cache tags, needs to be updated or if data needs to be provided. Performance of such snoop operations may add to latency and processing overhead in computer systems.

SUMMARY

[0011] Various embodiments of a mechanism for filtering snoop requests to a cache memory are disclosed. In one embodiment, the mechanism may include a storage including a plurality of entries configured to store corresponding snoop filter indications. The mechanism may also include a cache controller configured to receive a transaction request including an address and to generate an index for accessing the storage by performing a hash function on the address. The cache controller may selectively generate a snoop operation to the cache memory for the transaction request dependent upon a snoop filter indication stored in the storage that corresponds to the address.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] FIG. 1 is a block diagram of a multiprocessing computer system employing a cache memory subsystem.

[0013] FIG. 2 is a block diagram of one embodiment of the cache memory subsystem of FIG. 1 including a snoop filter employing a hash function.

[0014] FIG. 3 is a block diagram of an alternative embodiment of the cache memory subsystem of FIG. 1 including a snoop filter employing a hash function.

[0015] FIG. 4 is a block diagram of another alternative embodiment of the cache memory subsystem of FIG. 1 including a snoop filter employing a plurality of hash functions.

[0016] While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the present invention as defined by the appended claims. Note, the headings are for organizational purposes only and are not meant to be used to limit or interpret the description or claims. Furthermore, note that the word "may" is used throughout this application in a permissive sense (i.e., having the potential to, being able to), not a mandatory sense (i.e., must). The term "include" and derivations thereof mean "including, but not limited to." The term "connected" means "directly or indirectly connected," and the term "coupled" means "directly or indirectly coupled."

DETAILED DESCRIPTION

[0017] Turning now to Figure 1, a block diagram of one embodiment of a computer system is shown. Computer system 10 includes a processor 100, a processor 150, a system device 160 and a system memory 170 interconnected through a system bus 140. Processor 100 illustratively includes a cache controller 110 that is coupled to a cache memory 120. Cache controller 110 and cache memory 120 are collectively referred to as a cache-memory subsystem 105. For simplicity, elements shown within the processor 100 are not shown within processor 150. However, it is noted that processor 150 may be configured identically to processor 100, as illustrated.

[0018] In the illustrated embodiment, processor 100 and cache controller 110 are integrated upon a common integrated circuit chip. However, it is contemplated that in other embodiments, processor 100, and cache controller 110 may be implemented upon separate integrated circuit chips. In one embodiment, processor 100 may be illustrative of a processor in the SPARC™ family of processors although other processor architectures are possible and contemplated.

[0019] In one embodiment, cache memory 120 may include a number of memory devices (not shown). The memory devices may be implemented using some type of fast memory devices such as static random access memory (SRAM) devices or dynamic RAM (DRAM) devices implemented as pseudo-SRAM devices, for example. During operation, cache memory 120 stores a plurality of cache lines. In one embodiment, each cache line may be stored in a row formed by corresponding entries of the memory devices. For example, a given cache line is stored across a set of locations of the memory devices that may be accessed using a common index address. In addition, a tag corresponding to the address may be stored along with the cache line.

[0020] In the illustrated embodiment, system device 160 is representative of any type of system device capable of performing coherent memory transactions. For example, system device 160 may be an I/O bridge device that may perform a read/modify/write operation to a piece of data within a block of memory of system memory 170. To perform such an operation, system device 160 may need to gain ownership of the memory block and depending on the coherency protocol, the corresponding cache line within cache memory 120.

[0021] In one embodiment, system memory 170 may include one or more memory devices. In addition, system memory 170 may include a memory controller (not shown) to control accesses to the memory devices. The memory devices may be implemented using any type of device in the dynamic random access memory (DRAM) family of devices such as synchronous DRAM (SDRAM) or double data rate SDRAM (DDRSDRAM), for example. It is noted that in one embodiment, the memory devices may be mounted to one or more memory modules (not shown).

[0022] Cache memory 120 is configured to store cache lines of data. Remote devices such as processor 150 or system device 160, for example, may perform transaction requests to cache memory 120 possibly resulting in snoop operations to cache memory 120. As will be described in greater detail below in conjunction with the description of FIG. 2 through FIG. 4, to reduce the number of snoop operations to cache memory 120, cache controller 110 may implement a snoop filter that selectively generates snoop operations to cache memory 120 when the requested cache line is present in cache memory 120.

[0023] Referring to FIG. 2, a block diagram of one embodiment of cache memory subsystem 105 including a snoop filter employing a hash function is shown. Components that correspond to those shown in FIG. 1 are numbered identically for clarity and

simplicity. Cache memory subsystem 105 includes a cache controller 110 coupled to a cache memory 120. Cache controller 110 is coupled to a storage 220. It is noted that in the illustrated embodiment, cache controller 110 and storage 220 are manufactured on the same integrated circuit device.

[0024] As described above, cache memory 120 is configured to store a plurality of cache lines each having a corresponding address. In addition, a tag (T) corresponding to the address may be stored along with the cache line data.

[0025] In the illustrated embodiment, cache controller 110 includes a snoop filter 205 configured to receive remote cache transaction requests that include an address from other devices such as processor 150 of FIG. 1, for example. In addition, cache controller 110 is configured to receive local cache transaction requests from core logic within processor 100. As snoop filter 205 receives each remote cache transaction request, a hash function 215 is performed on the address associated with the remote cache transaction request. In addition, hash function 215 is also performed on the address associated with some local cache transaction requests such as write requests, for example. The resultant hash function value corresponds to an index for accessing storage 220.

[0026] In the illustrated embodiment, storage 220 includes a plurality of locations. In one embodiment, each location stores a snoop filter indication (SI) and may be accessed by a corresponding index value. For example, storage 220 may include 1000 locations, numbered 0 through 999. In one embodiment, the index values may be 0 through 999. As will be described further below, in one embodiment, a snoop filter indication may be a single bit indicative of whether a given cache line is not present within cache memory 120. It is noted that in one embodiment, storage 220 may include a greater number of locations than cache memory 120.

[0027] In one embodiment, cache subsystem 200 may be initialized during an initialization of processor 100, for example. In such an embodiment, since cache memory 120 may be empty after such an initialization, each location of storage 220 may be initialized to store a snoop filter indication indicating that the corresponding cache line is not present within cache memory 120. In one embodiment, the snoop filter indication may be a single bit having a logic value of zero, for example, to indicate that a corresponding cache line is not present within cache memory 120. It is noted however, that other embodiments may use other numbers of bits and /or other logic values to indicate that the corresponding cache line is not present within cache memory 120.

[0028] During operation, the core logic of processor 100 sends local cache transaction requests to cache controller 110 to write lines of data to cache memory 120. Cache controller 110 is configured to store the lines of data to cache memory 120. In one implementation, snoop filter 205 performs hash function 215 on the address associated with each local cache transaction request to generate a hash function value. In addition, cache controller 110 is configured to store a snoop filter indication within the storage location of storage 220 that is indexed by the hash function value. In one embodiment, the snoop filter indication may be a single bit having a logic value of one to indicate that a corresponding cache line may be present within cache memory 120. It is noted however, that other embodiments may use other numbers of bits and /or other logic values to indicate that the corresponding cache line may be present within cache memory 120.

[0029] In addition, in one embodiment, snoop filter 205 is configured to filter any snoops to cache memory 120 by checking storage 220 prior to performing a snoop of cache memory 120. For example, in response to receiving a remote cache transaction request, snoop filter 205 is configured to generate a hash function value by performing a hash function on the address corresponding to a remote cache transaction request. Snoop

filter 205 reads the snoop filter indication stored in the location within storage 220 indexed by the hash function value. If the snoop filter indication indicates that the cache line is not stored within cache memory 120, snoop filter 205 causes cache controller 110 to ignore the remote cache transaction request, rather than generating an associated snoop operation to cache memory 120. However, if the snoop filter indication indicates that the cache line may be stored within cache memory 120, cache controller 110 is configured to perform a snoop of the cache memory 120 for the remote cache transaction request.

[0030] Accordingly, by filtering snoops to cache memory 120, valuable bandwidth that is normally used for remote snoop transactions may be used instead for local processor transactions. However, since hash function 215 may generate the same hash function value and hence the same index for a number of different addresses within a given address range, a given snoop filter indication may only positively identify that a corresponding cache line of data is not stored within cache memory 120. In addition, as cache lines get evicted and new cache lines are stored, the snoop filter indications stored within storage 220 may become outdated and therefore erroneous. Further, as storage 220 becomes full of snoop filter indications that indicate that cache lines may be present within cache memory 120, the chance of falsely identifying a cache line as being present increases, which may make the effectiveness of snoop filter 205 decrease.

[0031] To minimize false cache line hits resulting from storage 220 filling with snoop filter indications indicating that particular cache lines may be present within cache memory 120, in one embodiment, cache controller 110 may temporarily disable snoop filter 205 from filtering snoops and a repopulation process of storage 220 may be performed. More particularly, during the repopulation process, cache controller 110 is configured to initialize each snoop filter indication to indicate that no cache lines are present within cache memory 120. In addition, cache controller 110 is configured to read each tag stored within cache memory 120. Snoop filter 205 is configured to perform hash

function 215 on each tag that is read. As each new hash value and corresponding index value is generated, cache controller 110 is configured to repopulate storage 220 with up-to-date snoop filter indications for the tags in cache memory 120. It is noted that in other embodiments, the repopulation process may be performed and repeated at some predetermined interval of time. Thus, the effectiveness of snoop filter 205 may be improved when storage 220 is repopulated with up-to-date snoop filter indications. It is noted that during the repopulation process, cache controller 110 may generate snoop operations for each remote transaction request that is received.

[0032] In an alternative embodiment, hash function 215 may be structured such that separate portions of cache 120 may map to respective portions of storage 220. For example, cache 120 may be divided into four portions and storage 220 may include four portions that correspond to the four cache portions. In such an embodiment, when a given portion of storage 220 begins to be filled and thus needs to be repopulated, snoop filtering may be disabled only for that portion of cache 120 that corresponds to the portion of storage 220 needing to be repopulated. Thus, during repopulation of that portion of storage 220, snoop filtering may continue for the remaining portions of cache 120. It is contemplated that cache 120 may be divided into any number of portions and storage 220 may include a corresponding number of portions.

[0033] Turning to FIG. 3, a block diagram of an alternative embodiment of cache memory subsystem 105 including a snoop filter employing a hash function is shown. Components that correspond to those shown in FIG. 1 are numbered identically for clarity and simplicity. It is noted that cache subsystem 105 of FIG. 3 includes all of the components and operates similar to cache subsystem 105 of FIG. 2. However, cache memory subsystem 105 of FIG. 3 additionally includes storage 330 and includes additional functionality as described below. As such, the operation of cache memory subsystem 105 of FIG. 3 that differs from the operation of cache memory subsystem 105

of FIG. 2 and/or is necessary for the understanding is described for simplicity. It is noted that in one embodiment, storage 330 is implemented on the same integrated circuit as cache controller 110.

[0034] In the illustrated embodiment, storage 330 is coupled to cache controller 110. Storage 330 may be substantially identical to storage 220. As described above in conjunction with the description of FIG. 2, the effectiveness of snoop filter 205 may decrease as storage 220 fills with snoop filter indications that are indicative that a cache line may be present within cache memory 120. Thus, to minimize false cache line hits, cache controller 110 is configured to stop using storage 220 and to use storage 330 after a population process of storage 330 is performed and storage 220 becomes full or substantially full.

[0035] Accordingly, in one embodiment, cache controller 110 is configured to operate using two modes of operation. For example, during a first mode of operation, cache controller 110 filters snoops using storage 220 until storage 220 begins to fill as described above. Cache controller 110 is also configured to populate storage 330 with up-to-date snoop filter indications using bandwidth not used by local cache transactions. During a second mode of operation, cache controller 110 filters snoops as described above, except that storage 330 is used instead of storage 220.

[0036] Specifically, in one embodiment, during the population of storage 330, cache controller 110 is configured to iteratively access each location of cache memory 120 and to read the tag of each valid entry stored therein. In addition, snoop filter 205 is configured to perform hash function 215 on each tag that is read. As each new hash value is generated, cache controller 110 is configured to access and populate storage 330 with an up-to-date snoop filter indication for the tag of each valid entry stored in cache memory 120.

[0037] Once the population of storage 330 is complete, cache controller 110 is configured to filter snoops using storage 330. Further, in one implementation, in response to cache controller 110 beginning to use storage 330, cache controller 110 may be further configured to initialize each location of storage 220 to have a snoop filter indication that is indicative that the corresponding cache line is not present within cache memory 120, as previously described above. In one embodiment, the population process may repeat continuously and cache controller 110 may cycle between storage 220 and storage 330.

[0038] Since hash function 215 may generate the same hash function value and hence the same index for a number of different addresses within a given address range, a given snoop filter indication may only positively identify that a corresponding cache line of data is not stored within cache memory 120. As will be described in greater detail below, to increase the effectiveness of filtering snoops and to minimize false hits, additional hash functions may be used to more accurately identify whether a cache line may be present within cache memory 120.

[0039] Referring to FIG. 4, a block diagram of another alternative embodiment of cache memory subsystem 105 including a snoop filter employing a plurality of hash functions is shown. Components that correspond to those shown in FIG. 1 are numbered identically for clarity and simplicity. Cache memory subsystem 105 of FIG. 4 includes a cache controller 110 coupled to a cache memory 120. Cache controller 110 is coupled to a storage 420 and a storage 430. It is further noted that in the illustrated embodiment, cache controller 110, storage 420 and storage 430 are manufactured on the same integrated circuit.

[0040] In the illustrated embodiment, cache controller includes a snoop filter 405 configured to receive remote cache transaction requests that include an address from other devices such as processor 150 of FIG. 1, for example. In addition, cache controller 110 is configured to receive local cache transaction requests from core logic within processor 100. As snoop filter 405 receives each remote cache transaction request, hash function 415A and hash function 415B are performed on the address associated with the remote cache transaction request. In addition, hash function 415A and hash function 415B are also performed on the address associated with some local cache transaction requests such as write requests, for example. Each resultant hash function value corresponds to an index for accessing storage 420 and storage 430, respectively.

[0041] In the illustrated embodiment, storage 420 and storage 430 each include a plurality of locations. In one embodiment, each location of storage 420 stores a first snoop filter indication (SI1) and as described above storage 420 may be accessed using a corresponding index. In addition, each location of storage 430 stores a second snoop filter indication (SI2) and as described above storage 430 may be accessed using a corresponding index. As will be described further below, in one embodiment, each snoop filter indication may be a single bit indicative of whether a given cache line is not present within cache memory 120. It is noted that in one embodiment, storage 420 and storage 430 may each include a greater number of locations than cache memory 120.

[0042] In one embodiment, cache subsystem 400 may be initialized during an initialization of processor 100, for example. In such an embodiment, since cache memory 120 may be empty after such an initialization, each location of storage 420 and storage 430 may be initialized to store a snoop filter indication indicating that the corresponding cache line is not present within cache memory 120. In one embodiment, each snoop filter indication may be a single bit having a logic value of zero, for example, to indicate that a corresponding cache line is not present within cache memory 120. It is noted however,

that other embodiments may use other numbers of bits and /or other logic values to indicate that the corresponding cache line is not present within cache memory 120.

[0043] During operation, the core logic of processor 100 sends local cache transaction requests to cache controller 110 to write lines of data to cache memory 120. Cache controller 110 is configured to store the cache of data to cache memory 120. In one embodiment, snoop filter 405 performs hash function 415A and 415B on the address associated with each local cache transaction request to generate a respective hash function value. In addition, cache controller 110 is configured to store a snoop filter indication within the storage locations of storage 420 and storage 430 indexed by the hash function values. In one embodiment, each snoop filter indication may be a single bit having a logic value of one to indicate that a corresponding cache line may be present within cache memory 120. It is noted however, that other embodiments may use other numbers of bits and /or other logic values to indicate that the corresponding cache line may be present within cache memory 120.

[0044] In addition, in one embodiment, snoop filter 405 is configured to filter any snoops to cache memory 120 by checking storage 420 and storage 430 prior to performing a snoop of cache memory 120. For example, in response to receiving a remote cache transaction request, snoop filter 405 is configured to generate hash function values by performing hash functions 415A and 415B on the address corresponding to a remote cache transaction request. Snoop filter 405 reads snoop filter indication SI1 stored in the location within storage 420 indexed by hash function value 415A. Snoop filter 405 also reads the snoop filter indication SI2 stored in the location within storage 430 indexed by hash function value 415B. In one implementation, if either snoop filter indications SI1 or SI2 indicates that the cache line is not stored within cache memory 120, snoop filter 405 is further configured to ignore the remote cache transaction request. However, if both snoop filter indications SI1 and SI2 indicate that the cache line may be

stored within cache memory 120, cache controller 110 is configured to perform a snoop operation to the cache memory 120 for the remote cache transaction request.

[0045] Similar to the descriptions of FIG. 2 and FIG. 3 above, as cache lines get evicted and new cache lines are stored, the snoop filter indications stored within storages 420 and 430 may become outdated and therefore erroneous. Further, as storages 420 and 430 become full of snoop filter indications that indicate that cache lines may be present within cache memory 120, the chance of falsely identifying a cache line as being present increases, which may make the effectiveness of snoop filter 405 decrease. Accordingly, in one embodiment, cache controller 110 may temporarily disable snoop filter 405 from filtering snoops and a repopulation process of storages 420 and 430 may be performed as described above in conjunction with the description of FIG. 2. As each tag is read from cache memory 120, hash functions 415A and 415B are used to repopulate storages 420 and 430, respectively.

[0046] Further, as described above in the description of FIG. 3, instead of temporarily disabling snoop filtering to repopulate the snoop filter storage, a second snoop filter storage may be employed. Similarly, it is contemplated that in another embodiment of cache subsystem 105 of FIG. 4, another pair of storages (not shown), which are substantially identical to storages 420 and 430, may be used to store snoop filter indications. In such an embodiment, cache controller 110 may alternately populate and switch between the sets of storages similar to the operation described in conjunction with the description of FIG. 3.

[0047] Although the embodiments above have been described in considerable detail, numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.